Polymorphism and Interface in Python

Assignment 3

2024

Agenda

- Introduction to OOP in Python
- What is Polymorphism?
- Polymorphism in Python
- Real-world Examples of Polymorphism
- Method Overriding
- Duck Typing
- Operator Overloading
- Interface Concept in Python
- Abstract Base Classes
- Implementing Interfaces in Python
- Practical Examples
- Benefits of Interfaces
- Polymorphism with Interfaces
- Summary and Conclusion

Introduction to OOP in Python

- Object-Oriented Programming (OOP) is a programming paradigm.
- It relies on the concepts of classes and objects.
- Key principles include: Encapsulation, Inheritance, Polymorphism, and Abstraction.

What is Polymorphism?

- Polymorphism allows objects of different classes to be treated as objects of a common super class.
- It provides the ability to redefine methods for derived classes.
- Literally means "many forms."

Polymorphism in Python

- Python supports polymorphism in several ways:
  - Method Overriding
  - Duck Typing
  - Operator Overloading

Method Overriding

- Method overriding allows a derived class to provide a specific implementation of a method defined in the base class.
- This is a key aspect of polymorphism.

Method Overriding Example

```
class Animal:
    def speak(self):
        print("Some sound")

class Dog(Animal):
    def speak(self):
        print("Bark")

class Cat(Animal):
    def speak(self):
        print("Meow")
```

Duck Typing

- Duck typing is a concept where the type of an object is determined by its behavior (methods and properties) rather than its inheritance.
- "If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck."

Duck Typing Example

```
class Bird:
    def fly(self):
        print("Bird is flying")

class Airplane:
    def fly(self):
        print("Airplane is flying")

def let_it_fly(obj):
    obj.fly()

let_it_fly(Bird())
let_it_fly(Airplane())
```

Operator Overloading

- Operator overloading allows different classes to define their behavior for operators like ‘+‘, ‘-‘, ‘*‘, etc.
- This is another form of polymorphism in Python.

Operator Overloading Example

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

p1 = Point(1, 2)
p2 = Point(3, 4)
result = p1 + p2
print(result.x, result.y)  # Output: 4, 6
```

Real-world Example of Polymorphism

- Different vehicles (e.g., car, bike, bus) can have a common method ‘start_engine()‘, but each implementation is unique.
- Promotes code reusability and simplifies maintenance.

Interface in Python

- Unlike some other languages, Python does not have built-in support for interfaces.
- Interfaces can be implemented using abstract base classes (ABCs).
- Defined in the ‘abc‘ module.

Abstract Base Classes (ABCs)

- Abstract Base Classes are used to define a set of methods that must be implemented in derived classes.
- ABCs provide a way to enforce certain methods in derived classes.

Abstract Base Class Example

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius
```

Benefits of Interfaces

- Provide a blueprint for classes.
- Facilitate multiple inheritance.
- Allow for consistency across different implementations.

Polymorphism with Interfaces

- Interfaces ensure that derived classes implement specific methods.
- Promotes flexibility and interchangeability of components.

Example: Polymorphism with Interfaces

```
class Bird(ABC):
    @abstractmethod
    def fly(self):
        pass

class Sparrow(Bird):
    def fly(self):
        print("Sparrow can fly high")

class Penguin(Bird):
    def fly(self):
        print("Penguin cannot fly")
```

Summary

- Polymorphism allows methods to have different implementations.
- Interfaces, implemented using ABCs, provide a contract for classes.
- Together, they help achieve flexibility, reusability, and maintainability in code.

Questions?

Feel free to ask any questions!

Conclusion

- Polymorphism and interfaces are crucial concepts in OOP.
- Python provides powerful mechanisms to implement these concepts in a simple manner.